ANNUAL

PROGRESS REPORT

RESEARCH

ON

AUTOMATIC CLASSIFICATION, INDEXING

AND EXTRACTING

F. T. Baker
M. Jones
G. L. Johnson
J. H. Williams

CONTRACT NONR 4456(00)

Submitted to

Information Systems Branch
Office of Naval Research
Department of the Navy
Washington, D. C., 20360

Prepared by

Federal Systems Division
International Business Machines Corporation
Gaithersburg, Maryland 20760

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Federal Systems Division International Business Machines Corp. Gaithersburg, Maryland 20760	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP

3. REPORT TITLE

RESEARCH ON AUTOMATIC CLASSIFICATION, INDEXING AND EXTRACTING

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Annual Progress Report

5. AUTHOR(S) (Last name, first name, initial)

Baker, F. T., Johnson, G. L., Jones, M., Williams, J. H.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
April 1966	42	

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
NONR 4456(00) b. PROJECT NO. c. d.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. AVAILABILITY/LIMITATION NOTICES

Qualified requesters may obtain copies of this report from DDC. Other qualified users shall request copies of this report from the originator.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Information Systems Branch Office of Naval Research Dept. of the Navy, Washington, D. C.

13. ABSTRACT In order to contribute to the success of several studies for automatic classification, indexing and extracting currently in progress, as well as to further our theoretical and practical understanding of textual item distributions, this year's funds under Contract No. NONR 4456(00) have been applied to the development of a frequency program capable of supplying these types of information. The program planned for the System/360, will provide numerous user options covering the format of the input text, the definition of a countable item (e.g., a "word" may be specified as any string of characters between delimiters such as comma, space, period, or any combination thereof), the definition of a textual unit over which frequencies are to be subtotaled (e.g., sentence, paragraph, or document), the types of data to be output, and the machine configuration to be used. Also, facility will be provided for the incorporation of user-supplied routines to perform special functions such as word pair generation, suffix normalization, etc.

In addition to the determination of program requirements and overall program design, progress has been made on the design of the Dictionary Build module of the frequency program. The main purpose of the program is the provision of an output containing an ordered list of the items, their frequencies, and any special tags desired by the user. For the processing of large input texts, efficient utilization of storage devices by rapid dictionary search and storage techniques were considered essential to the complete program. The Dictionary Build module is therefore a critical one and has received special attention.

Contained in this report are descriptions of the requirements generated for the System/360 Frequency Program, status report on program design and documentation of dictionary construction methods.

DD FORM 1 JAN 64 1473

Security Classification

UNCLASSIFIED

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Frequency Program Textual Item Counting Program Design Dictionary Processing						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2a. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7a. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8a. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through _____."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through _____."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through _____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS)*, *(S)*, *(C)*, or *(U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.

Security Classification

TABLE OF CONTENTS

Section			Page
1	INTRODUCTION		1
2	PROGRAM REQUIREMENTS		4
	2.0	Introduction	4
	2.1	Machine Configuration	4
	2.2	Input Specifications	5
	2.3	Item Definition	5
	2.4	Interval Specification	6
	2.5	Special Tagging	6
	2.6	Output Requirements	6
		2.6.1 Concordance	7
		2.6.2 Summary Frequency Data	7
		2.6.3 Detailed Frequency Data	7
		2.6.4 Growth Rate Data	8
	2.7	Programming	8
	2.8	Operation	8
3	PROGRAM DESIGN		9
	3.0	Introduction	9
	3.1	Modular Approach	9
	3.2	Control Program	10
	3.3	Basic Modules	11
		3.3.1 Input Module	11
		3.3.2 Item Identification Module	11

CONTENTS (cont'd.)

Section				Page
3		3.3.3	Dictionary Build Module	12
		3.3.4	Merge Module	12
		3.3.5	Detailed Output Module	13
	3.4	Program - Provided Option Modules		13
		3.4.1	Concordance Module	13
		3.4.2	Special Item Check Module	14
		3.4.3	Growth Rate Module	14
		3.4.4	Summary Output Module	15
	3.5	User - Provided Optional Modules		15
	3.6	Examples		16
		3.6.1	Article Separation & Analysis	16
		3.6.2	Multi-Level Classification	17
		3.6.3	Trigram & Tetragram Analysis	17
4	DICTIONARY PROCESSING			23
	4.0	Introduction		23
	4.1	Overall Data Flow		24
		4.1.1	Overall Sort/Merge	24
		4.1.2	Input Iteration	25
		4.1.3	Partial Dictionary Generation	26
		4.1.4	Analysis	27
	4.2	Dictionary Construction Techniques		27
		4.2.1	Straight Chain Method	28
		4.2.2	Binary Search Method	31
		4.2.3	Directory-Chain Method	33
5	GLOSSARY			41

LIST OF ILLUSTRATIONS

Figure		Page
3-1	Article Separation and Analysis	19
3-2	Multi-Level Classification	20
3-3	Trigram and Tetragram Analysis	21
3-4	Modules by Type	22
4-1	Chaining Tables	30
4-2	Binary Search	32

Section 1

INTRODUCTION

Language analysis studies are necessary prerequisites to successful establishment and operation of language data processing systems. For example, it is usually necessary to determine the characteristics of input text and queries prior to the development of techniques for document processing. Language analysis studies can provide information on syntax, semantics, word counts, patterns, associations, etc., which is valuable in the development of techniques for translation, classification, indexing, abstracting, correction, structuring, prediction, etc.

In support of these language analysis studies, particularly in the areas of automatic classification, indexing and extracting, IBM has been engaged in research on the statistical and morphological behavior of character strings or items in narrative text. The present IBM technique for automatic document classification requires word counts by individual document and by document category. Work on word morphology currently being performed uses counts of syllables, n-grams (items n characters in length) and positional distribution of individual characters. Morphological analysis of words provides statistics necessary for automatic methods of hyphenation-justification, index term selection, and textual error correction. Automatic extracting methods are partially based upon information from word counts by sentence, paragraph, and document. Finally, the statistics generated by processing samples of input text and queries can yield the values of several document system design parameters; a few of these are:

expected dictionary size and growth rate, magnitude of the input error problem, expected document length and frequency of special types of words such as proper nouns.

In order to contribute to the success of these several studies currently in progress, as well as to further our theoretical and practical understanding of textual item distributions, this year's funds under Contract No. NONR 4456(00) have been applied to the development of a frequency program capable of supplying these types of information. The program planned for the System/360, will provide numerous user options covering the format of the input text, the definition of a countable item (e.g., a "word" may be specified as any string of characters between delimiters such as comma, space, period, or any combination thereof), the definition of a textual unit over which frequencies are to be subtotaled (e.g., sentence, paragraph, or document), the types of data to be output, and the machine configuration to be used. Also, facility will be provided for the incorporation of user-supplied routines to perform special functions such as word pair generation, suffix normalization, etc.

In addition to the determination of program requirements and overall program design, progress has been made on the design of the Dictionary Build module of the frequency program. The main purpose of the program is the provision of an output containing an ordered list of the items, their frequencies, and any special tags desired by the user. For the processing of large input texts, efficient utilization of storage devices by rapid dictionary search and storage techniques were considered essential to the complete program. The Dictionary Build module is therefore a critical one and has received special attention.

Contained in this report are descriptions of the requirements generated for the System/360 Frequency Program, status report on program design and documentation of the dictionary construction methods which have been studied for possible use. Evaluation of the dictionary construction methods is continuing, and efficiency comparisons in this area

will be obtained through experimental calculations or actual programmed testing. Design of other program modules will continue, and programming will begin upon completion of the design work.

Section 2

PROGRAM REQUIREMENTS

2.0 INTRODUCTION

Before commencing the design of a general frequency program, a study of the capabilities required and options desired was made. A general frequency program for the IBM 7090 has been in use for four years. Over this period it has been rewritten once and modified several times to adapt it to an ever-increasing number of applications. In addition, other programs in use which perform similar functions were studied. The capabilities and desirable features described below were selected as a result of this study.

2.1 MACHINE CONFIGURATION

The program should be planned for the IBM System/360, which comprises a compatible set of central processors and auxiliary units. The program should be written to be operable on one of the smaller models of System/360 and should take advantage of extra facilities when run on larger models.

The general organization of a frequency program requires the repetitive building of lists of items and frequency data followed by merging groups of these lists. A reasonable amount of core storage must be available for list generation and merging purposes. Also, for list storage and merging purposes, direct access auxiliary storage is highly desirable. Hence, the minimum configuration for the program will be a

System/360 with sufficient core storage, direct access storage units such as the IBM 2311, IBM 2314 or IBM 2302 disk storages or the IBM 2321 data cell drive for list storage and merging. Direct access or serial access storage devices such as the IBM 2400 series tape drives will be used to handle input and output requirements. The exact amounts of storage and type and number of devices required will be determined in the detailed program design process.

2.2 INPUT SPECIFICATIONS

Inputs to the program can consist of narrative text or a variety of types of formatted records. The program should be capable of reading input in serial fashion from either serial access or direct access devices. It should allow for variable as well as fixed length records up to a reasonable maximum length. The standard character code should be EBCDIC. However, the program should allow the user to convert to or from non-standard character codes.

2.3 ITEM DEFINITION

The program should be capable of providing data on the frequency of a number of types of items. Once the items have been identified in the input, this process is relatively straightforward and common to all types. However, the item identification process can vary widely according to the form of the input and the outputs desired. The most frequent use of the program is expected to be on textual data with word frequencies desired. Hence the program should provide a word identification routine to operate on narrative text. The specification of "words" should be determined by the user at the time he sets up a run and should allow for a variety of word delimiting characters (e.g., blank, comma, period, hyphen, single quote) and rules for applying them. The program should also allow for provision by the user of other types of item identification

routines to be incorporated in place of the standard routine. Such routines could perform word pair definition, individual character breakup, word encoding, matching of syllables against a dictionary or other desired item definition functions.

2.4 INTERVAL SPECIFICATION

It should be possible to specify several textual intervals at which frequencies should be tallied. For example, one may desire word frequencies by sentence, paragraph and document from the same set of input text. The variety of possible input formats makes it impractical to provide standard modules to accomplish interval determination. The standard program should therefore provide for multiple intervals of tally, and the user should provide a routine for interval determination if he desires this function. If no such routine is provided, the program should tally frequencies over the entire input.

2.5 SPECIAL TAGGING

It is desirable in some cases to allow tagging of various types of items. One may wish to separate numeric items from alphanumeric or alphabetic ones, identify special items in some way, or tag the same item in different ways according to its context. The program should provide for a user-inserted special routine to perform any tagging desired and should take such tags into account in building and merging lists of items and their frequencies.

2.6 OUTPUT REQUIREMENTS

Many types of outputs have been found desirable from frequency programs. These generally fall into four categories, each of which is discussed separately below.

2.6.1 Concordance

For some purposes it is necessary to retain position information about items. This can then be sorted or processed in a variety of ways to get data on the way these items are used in context. The program should provide an optional concordance output containing, for each identified item, or for a pre-specified list of items:

a. Item
b. User-supplied tags
c. Interval identification
d. Sequential position within interval

2.6.2 Summary Frequency Data

A number of summary outputs are useful in giving an overall view of the data. The program should provide, at each interval specified:

a. Item type-token distribution and total number of types and tokens
b. Distribution of item types by initial character
c. Distribution of item tokens by initial character
d. Distribution of item types by string length
e. Distribution of item tokens by string length

2.6.3 Detailed Frequency Data

The desired basic output of the program is detailed information on individual items and their frequencies. This output might be sorted in a variety of ways by standard sort programs to group items by length, frequency, tags, etc. The standard program output should be:

a. Item
b. Tags
c. Frequency
d. Interval and sequence number of first occurrence

The standard output sort should be alphabetic by item within tags.

2.6.4 Growth Rate Data

In addition to the detailed data, information on the growth rate of the number of items in a vocabulary is frequently useful. This may be desired either at fixed intervals, such as every 5000 items, or at the user-specified textual intervals. The program should therefore provide an optional output of the number of items encountered either in the pre-specified textual intervals or at fixed intervals.

2.7 PROGRAMMING

In order to simplify modification and allow easy incorporation of user-provided routines, the basic program should be modular and should be programmed in a higher-level language. The use of PL/I should be considered, since it is an advanced language and appears to provide the necessary features.

2.8 OPERATION

The program should operate under Operating System/360. All input and output should be performed by standard OS/360 routines. Since the program may run for long periods when large amounts of input are provided, an option to stop the program, save necessary parameters and restart at a later time should be incorporated.

Section 3

PROGRAM DESIGN

3.0 INTRODUCTION

This frequency program is required to provide frequency data on a variety of types of items for many different purposes. Since it is impractical to try to anticipate all the uses, data formats, output requirements, etc., the design philosophy has been to provide a set of general-purpose program modules to perform a group of basic functions associated with building a list of items and their frequencies. The user can then supply any additional modules required for special-purpose operations and unusual input-output formats. A control program to select appropriate modules, assemble them into a working package and initiate and monitor the run will also be provided. Selection of a variety of summary outputs formatted for printing will be possible. Item frequency and concordance outputs will be provided in a format suitable for sorting or processing either by standard System/360 packages such as the Sort or Report Program Generator or by special-purpose programs provided by the user.

3.1 MODULAR APPROACH

Since the requirements for a given run may vary widely, the program has been broken into modules performing specific functions. Each module will have an initialization phase which sets up the module prior to processing. Thus, once the user specifies his options the module will adjust itself to operate efficiently in satisfying his requirements. The modules perform two types of functions, basic and optional.

functions.

A basic module is one which performs a function that is necessary to the building of a list of items and their frequencies from the textual input. They include input, item identification, dictionary building, merging and output modules.

The optional modules are of two types, the first provided by the frequency program, and the second provided by the user.

Program-provided optional modules are those which perform functions that are independent of the particular data being processed. The bulk of these consist of summarizing functions which can be selected to provide specific types of outputs.

The user must provide modules to perform functions which depend on variations in the format of the input data or present special processing or output requirements. An example of this type of function is encoding of the words. Instead of programming one encoding algorithm or even several algorithms which may not be applicable to this user's problem, the user can insert his own algorithm with only a minimum amount of knowledge about the entire frequency program. Provision has been made to permit the user to program modules of this type and insert them at the appropriate time.

A list of the modules of each type is given in Figure 3-4, and descriptions of them are contained in Sections 3.3, 3.4, and 3.5.

3.2 CONTROL PROGRAM

The modules are tied together through the control program, which consists of two phases. Phase One is devoted to the processing of the input parameters and the initialization of the modules to be used in the operational phase. Phase Two checks the options and assembles the operational program in the desired sequence; following the assembly, control is transferred to the operational program and the data processing begun.

3.3 BASIC MODULES

This section describes the modules which will be provided to perform the five required basic functions. These are input, item identification, dictionary building, merging and output of the detailed frequency data.

3.3.1 Input Module

The function of the Input module is to read the data to be frequency counted from the input medium. This data will be read using the standard System/360 data access methods, which will permit the module to remain device-independent. The initialization phase of this module will accept parameters describing the particular format of the data and will set up buffers to hold the data. Output of this module will be one logical record.

3.3.2 Item Identification Module

The function of the Item Identification module is the analysis of the input in order to identify the items to be counted. Every run of the frequency program must have such a module; however the particular module used will depend on the type of item the user wants counted.

A very frequent use of the frequency program will be to provide information on words occurring in narrative text. For this reason, a standard Item Identification module is provided. Modules for other types of item identification may be written either to operate on the output of the standard Item Identification module (e.g., for word pairs or syllable identification) or to replace it completely (e.g., for individual character identification).

The purpose of the standard Item Identfication module is to identify individual words in the input stream. The module accepts a set of characters which are legal characters for the word definition; that is, a word must consist of letters, digits or certain special characters from the

legal set. It also accepts a list of characters which serve as delimiters; together with a set of rules, these characters indicate the end of a word. The user may also define a list of characters which will cause the end of a word if not followed by a numeric character.

As an example, consider the use of the comma (,) in text. If the number 1,071 appears, a user would wish it to be stored in the dictionary as 1,071. On the other hand, if the phrase "cats, dogs, and mice" appears, the user would like the words "cats" and "dogs" to be entered into the dictionary without the comma. This can be accomplished by specifying the comma (,) as a character to be ignored and the blank or space as a delimiter character.

3.3.3 Dictionary Build Module

The Dictionary Build Module is the most critical module in the system since it has the greatest effect on efficiency. Consequently, methods of building dictionaries are being studied in greater detail, and Section 4 of this report discusses some of these methods.

The Dictionary Build module will accept an item as defined by Item Identification and/or user-supplied modules and search the dictionary for this item. Each item not found in the dictionary is inserted with a frequency count of one, and each item already in the dictionary has its frequency count augmented by one. When the interval over which the dictionary is being built changes, or when new items can no longer be added to the dictionary, the dictionary is placed on a disk in sorted order for later use in the Merge module. Provision is made in the dictionary format for multiple intervals and for any user-supplied tags.

3.3.4 Merge Module

The output of sorted partial dictionaries make the function of the Merge module quite simple. Merge will add the frequencies of duplicated items and output the composite dictionary back onto the disk. Since the

frequency program will allow the user to obtain outputs at any of the intervals desired, this Merge module will be required to combine dictionaries at the desired intervals for each of these outputs. For example, suppose there are three text intervals, A, B and C (e.g., data base, category, document) where A is the highest interval with units a_1, a_2, ... a_h and intervals B and C consist of units b_1, b_2, ..., b_n and c_1, c_2, ..., c_m respectively. The Merge module would first combine all partial dictionaries for $a_1b_1c_1$, $a_1b_1c_2$, ..., $a_1b_1c_m$, $a_1b_2c_1$, ..., $a_1b_2c_m$, ..., $a_1b_nc_m$, $a_2b_1c_1$, ..., $a_hb_nc_m$. Information would then be output for each $a_ib_jc_k$, where $i = 1, 2, \ldots, h$; $j = 1, 2, \ldots, n$; $k = 1, 2, \ldots, m$. The next Merge pass would merge all the dictionaries for each a_ib_j prior to their output. In the final Merge pass all dictionaries would be merged for the composite dictionary at interval A.

3.3.5 Detailed Output Module

The Detailed Output module reads the appropriate dictionary from disk and formats the items, their frequencies and any desired tags for printing. This is the module which provides the basic output in alphabetic order and will be used at each of the intervals specified. The method of output will be one of the standard System/360 methods which will permit the module to be device-independent.

3.4 PROGRAM-PROVIDED OPTIONAL MODULES

This section describes those program-provided optional modules which may be included in an operational program to perform special functions or provide additional outputs.

3.4.1 Concordance Module

The Concordance module provides a detailed record of the position of every item identified by the program. It allows users to analyze the

context of specified items, to note the location of certain items or to provide the data required for a full-text index. This module will output the item, any user-provided tags, the interval tag and the position of the item within the interval. This output will be performed using one of the standard System/360 access methods so that it will be device-independent. The user can then sort or otherwise process this data using other System/360 programs such as the Sort or Report Program Generator.

3.4.2 Special Item Check Module

The Special Item Check module permits the user to check an item defined by an item identification module against a list of items predetermined by the user. When an item on the list is found control will be given to a module provided by the user. This facility is provided for use in special tagging or interval analysis routines. For example, the beginning or end of a document may be identified by a special item. When an item of this type is encountered, the user can have control transferred to a module which changes an interval indicator so that succeeding items passed along to the Dictionary Build module will be tagged as belonging to a new interval.

3.4.3 Growth Rate Module

The Growth Rate module will keep a record of the unique items added to the dictionary over intervals specified by the user. After all input has been processed the growth information will be written onto an output device using one of the standard access methods. Output of this module can be used to determine the expected rate of additions to the vocabulary of a document collection, to provide clues to the location of error bursts in input data or to indicate when changes of subject have occurred.

3.4.4 Summary Output Module

The Summary Output module will consist of the following sub-modules: Token-Type, Frequency Distribution, Initial Character Distribution, Word Length Distribution. These sub-modules will be used as requested to form the summary output desired at each interval. The Token-Type sub-module lists the total number of tokens and types encountered. The Frequency Distribution sub-module gives the number of types which occurred with a frequency of one, the number of types which occurred with a frequency of two, etc. Initial Character Distribution gives the frequencies by initial character of both types and tokens. Finally, the Length Distribution sub-module gives the frequencies by item length for both types and tokens. The output is formatted for printing and processed by one of the standard access methods of System/360.

3.5 USER-PROVIDED OPTIONAL MODULES

User-provided modules will perform such functions as defining input intervals, tagging items, and encoding which are specially dependent on text content or format. For interval definition the user can examine the items defined by an Item Identification module. The user can then check this item to determine if it represents an interval identifier instead of an item. If more information is needed he can call the Input module which accesses the input data. The user can also tag items as numeric, alphabetic, alphanumeric, or whatever he wants identified in a particular manner. This tag can be placed either preceding or following an item depending on how the user desires his output to be sorted. Encoding can also be performed after an item has been defined. Since output is in sorted order a user might want to encode for the concordance only. In this case he could call the Concordance module from his encoding module and let the Dictionary module store items without encoding.

3.6 EXAMPLES

This section will present three examples drawn from actual experience for which several existing IBM 7090 frequency programs and some special-purpose programs were required. It will show how the System/360 program will be assembled and used to provide the data required by each of these problems.

3.6.1 Article Separation and Analysis

This problem was presented in the course of studying a large body of text from magazine articles for purposes of thesaurus development and query analysis. Requirements were for word frequency data both for total text and by individual articles, growth rate data keyed to individual articles so that major vocabulary changes and error bursts could easily be located, and summary information required to assist in determining the best organization for the thesaurus.

The articles were contained on paper tape, one article per strip, and were too numerous to be strip-fed into a reader. The strips were therefore spliced into reels and converted to magnetic tape. However, this destroyed the separation between articles, and a special program had to be written to analyze the text, search for article breakpoints and insert article marker strings. Two frequency programs were then run, one to get data for the total text, and a second to get data for individual articles.

The System/360 program could have handled this job with the addition of a single user-supplied module. This module would be called whenever a carriage return symbol was encountered by the Special String Check module and would determine whether this was a normal single carriage return or t' e beginning of a string of carriage returns which signified the end of an article. In the latter case an interval indicator would be changed so that the succeeding words would be identified as part of a new article. The user would supply this module and by control

cards request the control program to set up and execute the program shown in Figure 3-1.

3.6.2 Multi-Level Classification

This problem arose during the development of a technique for subject classification of documents into a hierarchic structure with several levels. The technique requires data on word frequencies at each level of the structure in order to generate classification parameters from a document sample. Sample documents are on cards with their identification and classification punched in columns 73-80. The present solution is to sort them by document within level and input them into a frequency program which produces word frequency data for each document. A special merge program is then used to provide information on the combined word frequencies within categories at each level of the structure.

The System/360 program provides all the routines necessary to perform this analysis with the exception of a module to check the identification and classification and provide interval tags for each hierarchic level. The user would supply this module and a set of control cards to establish the program shown in Figure 3-2.

3.6.3 Trigram and Tetragram Analysis

In the course of some word morphology work oriented toward automatic identification of proper nouns in teletype material without type-case symbols, a set of trigram and tetragram frequencies from a sample of proper nouns was desired. The solution was to run an existing 7090 word frequency program once to get word frequency data and then to input this data to a modified version of the same program which performed the gram identification function.

The System/360 program could perform the entire job in one pass with two user-supplied modules. The first module would be an Item Identi-

fication module which would extract only capitalized items, excluding words which begin a sentence. These items would be processed by the second user-supplied module which would repetitively form all its constituent trigrams and tetragrams. These would be tagged as one or the other and passed on to the Dictionary Build module, which in this case would be set up to return to the user-supplied module until the last gram had been added. The setup for this run is shown in Figure 3-3.

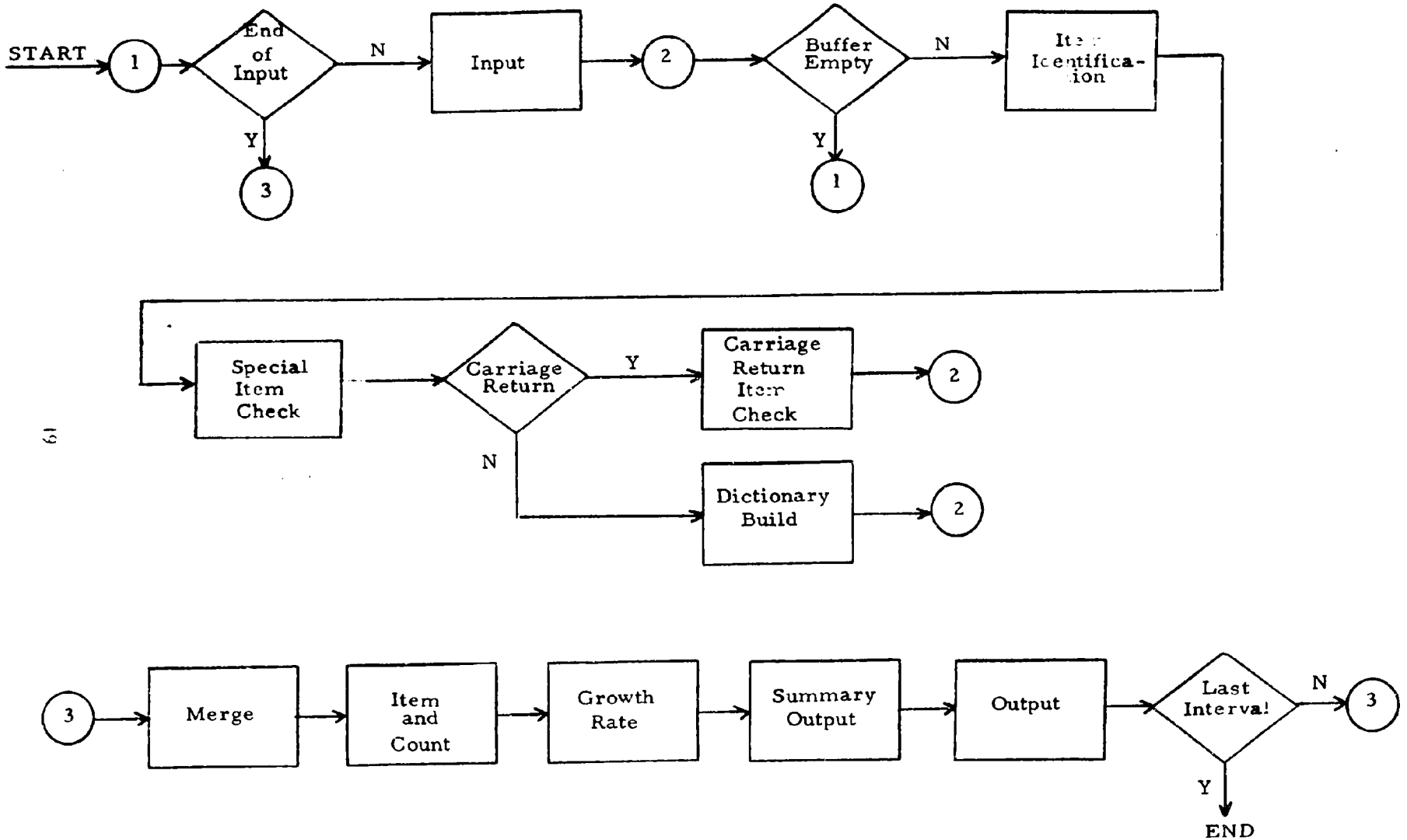


Figure 3-1

Article Separation and Analysis

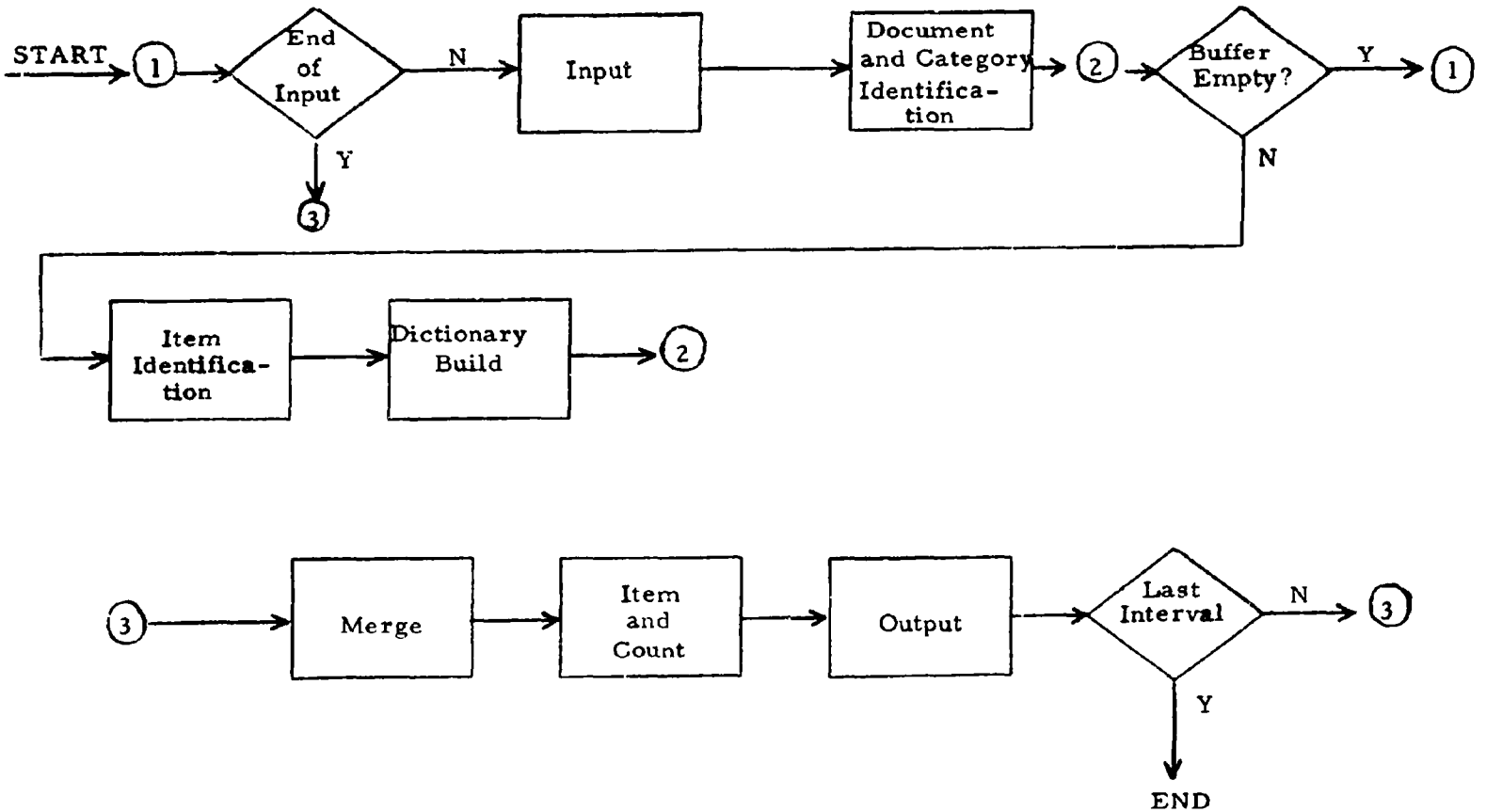


Figure 3-2

Multi-Level Classification

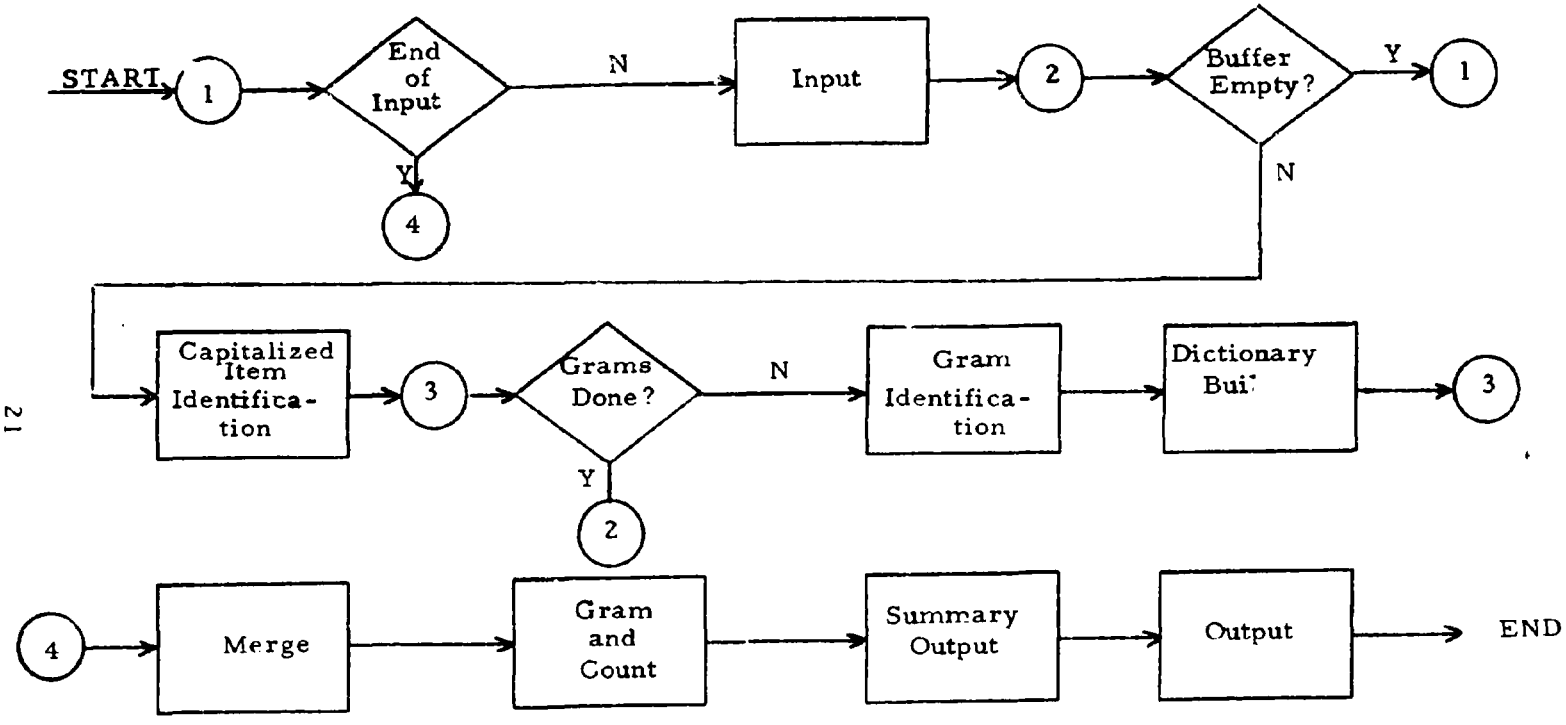


Figure 3-3

Trigram and Tetragram Analysis

Basic Modules

Input
Item Identification
Dictionary Build
Merge
Detailed Output

Program-Provided Optional Modules

Concordance
Special Item Check
Growth Rate
Summary Output
    Token-Type
    Frequency Distribution
    Initial Character Distribution
    Word Length Distribution

Possible User-Provided Optional Modules

Pre-processing of the Data
Interval Definition
Encoding
Word Tagging
Special Action on Special Words

Figure 3-4

Modules by Type

Section 4

DICTIONARY PROCESSING

4.0 INTRODUCTION

Some of the most important decisions which must be made in the course of the development of the frequency program are concerned with the manner in which partial dictionaries are generated and merged. In general, the number of item types in the input will be larger than can be held at once in core storage. Counts must therefore be made for separate segments of the input, and these paratial dictionaries later must be merged to provide final counts for each of the desired intervals. There are several basic approaches to organizing this overall data flow, and the efficiency of the method chosen is critical to overall program efficiency. Also, in the generation of the partial dictionaries, there is a variety of ways to organize and use core storage. The amount of time spent by the program on this task will be quite substantial, and here again a wise choice of method can pay dividends in operating efficiency.

In these two areas, then, special study of the possible approaches and their advantages and disadvantages is being made. The study of dictionary generation methods is not yet complete, and the decision on which of the several approaches to use will, as in the study of data flow methods, be based on quantitative estimates of time derived from experience with large text samples and previous programs, as well as on the simplicity of design and programming expected to be associated with each. The remainder of this section describes the various approaches which are

being evaluated, both for the overall data flow and for the construction of the individual partial dictionaries. It also outlines the evaluation performed on data flow techniques to select the method to be used in the program.

4.1 OVERALL DATA FLOW

The three different approaches to overall data flow which were considered for this program are Overall Sort/Merge, Input Iteration and Partial Dictionary Generation. These three methods and their variations are described below.

4.1.1 Overall Sort/Merge

This is perhaps the simplest conceptual method to the organization of the overall data flow. There are two variations, one of which is slightly more sophisticated.

The first and most straightforward way is to output each item and its sequence immediately after its identification in the input. When the entire input has been processed, a standard sort/merge can be performed on this data with a major sort on the item and a minor sort on sequence. At the completion of the merge, the items will be ordered alphabetically, and duplicates will be ordered in the sequence of their occurrence in text. This list can be processed as many times as required to get frequency data for each interval. Duplicates can be eliminated within each interval at each stage, reducing the amount of input to the succeeding stage.

A more sophisticated version of this process is to eliminate duplicate items during the merge process itself. Because of the fact that about 200 common words account for about half the words appearing in narrative English text, this process can reduce the merge time significantly. At the completion of the merge, the counting can proceed as in the first variation.

4.1.2 Input Iteration

This organization of overall data flow accepts input generated by an Item Identification module and creates a dictionary of unique items with their position values and frequencies, until the working storage area is filled. Any succeeding items appearing in the input but not appearing in the dictionary are saved on an intermediate storage unit. The positional values for the "overflow" input items are part of the data included on the temporary storage device. As many intermediate storage units are used as necessary to contain those input items not contained in the initial dictionary. At the end of processing the input items, the initial dictionary is saved on intermediate storage, and the overflow input items are reprocessed to form one or more additional dictionaries. At the completion of dictionary generation, these partial dictionaries are merged to form the final output dictionary of unique items with their frequency and position values.

In this iterative input processing method of dictionary construction, one of the problems to be considered is the limitation of storage capacity. It is possible to overcome some of the storage capacity limitations by allocating the storage area into various blocked sizes. The size of the blocks is predetermined based on the probabilistic occurrence of items by initial character. For example, in narrative text the words beginning with S are more frequent than words beginning with X and the dictionary block sizes would reflect this probability.

In this method of dictionary construction, as in others to be described, the type-token distribution of the input text is the principle controlling factor for causing multiple intermediate dictionaries to be created. The greater the redundancy of types in the input text, the less time, intermediate storage devices, and sort/merge operations will be required to process the tokens.

4.1.3 Partial Dictionary Generation

There are several alternative techniques available for organization of overall data flow by partial dictionary generation. The variations attempt to improve operational efficiency, by utilizing knowledge of certain characteristics of the input text. Three variations of this technique are briefly described.

The basic version of this technique accepts as input the items generated by an Item Identification module and the positional sequence values which have been assigned to each item. A dictionary of these unique items and their frequencies is generated until the working storage area is filled. This dictionary is saved on an intermediate storage unit, and the generation of a new dictionary of input items is begun. This partial dictionary build and save cycle is repeated until all the input text items have been processed. The partial dictionaries are merged, eliminating the duplicates, and the final dictionary contains the unique text items with frequency counts and positional sequence values.

This second variation also accepts text items with sequence values which have been provided by an Item Identification module. A dictionary of these unique items is generated until the working storage area is filled. At this time, those items in the storage area with the lowest frequencies are saved on an intermediate storage unit. Those items with the highest frequencies are retained in the working storage area, and additional input items are processed against these unique items to augment the dictionary. This cycle is repeated until all the input items have been processed. The resulting partial dictionaries are merged, eliminating duplicates and retaining the proper frequency counts and sequence values.

This third variation requires analysis of the type of input to be processed in order to construct an efficient pre-stored dictionary. The items with positional sequence values from an Item Identification module are compared against the pre-stored dictionary, and the dictionary

is augmented with any new input items until the working storage area is exhausted. Any of the pre-stored items which have not been found as an input item are eliminated at this time, and dictionary augmentation continues. When the working storage area is again filled, a partial dictionary of those items not occurring in the pre-stored dictionary is saved on an intermediate storage device. The cycle is repeated until all the input has been processed. The intermediate dictionaries are merged, eliminating duplicates and retaining the proper frequency counts and position values.

4.1.4 Analysis

Once the above approaches had been identified and described, an analysis was made to determine their relative effectiveness. The approach used was to estimate the amount of computer time used by each of the methods in acquiring an alphabetic list of the words and their frequencies encountered in a textual sample on which extensive data was available.

Preliminary analysis showed the Overall Sort/Merge methods to be non-competitive, so estimates were not obtained for this method. Based on estimates of machine cycles for compare/input/output units, the Partial Dictionary Generation methods are slightly superior to the Input Iteration method. However, the variation in performance between them is relatively insignificant, and it was decided that the specific variation chosen should depend on estimates of design and programming simplicity. The basic data flow will therefore follow the Partial Dictionary Generation organization, with the final specification awaiting the completion of the study of dictionary construction techniques.

4.2 DICTIONARY CONSTRUCTION TECHNIQUES

Fundamental to the efficiency of any of the overall data flow methods described are the techniques for dictionary construction.

Three basic techniques for dictionary construction are being investigated for this study: Straight Chain Method, Binary Search Method, and Directory-Chain Method. Placing a new item in the dictionary and searching for an item already in the dictionary require the same initial process, since before an item can be stored in the dictionary it is first necessary to determine whether or not it is already present. Therefore, these techniques can be described for either search or store operations with the functions being implicit.

4.2.1 Straight Chain Method

A chain can be used as a means of connecting non-contiguous items of arbitrary size. The primary advantages of the chaining technique in dictionary searching are: efficient utilization of available storage, rapid internal processing rate, and ease with which chains may be altered to revise inter-item relationships without item movement in storage.

The chain method requires a technique for converting the input item to a storage address. This address begins a chain of table entries, each containing the address of a dictionary entry. The chain is necessary since more than one entry may be converted to the same address, while there may exist addresses to which no entry converts. The method for generating an address for the input item may be algorithmic, may make use of an index or may be some combination of the two.

The following example shows how the chain can be generated, how the final dictionary would look, and how an item would be located. In this simple example an index is used to convert the first character of an item to table addresses such that:

A converts to 1
C converts to 3
D converts to 5
E converts to 7

Prior to beginning operation, all the other table entries are chained as available storage. An asterisk (*) is used to indicate the end of each chain in the table. Assume that sixteen words are to be stored, and that the range of addresses available is 1 through 16. The first word encountered beginning with the letter "A" is stored in the next available space in the dictionary, and that dictionary address is stored in table location 1. The table entries for succeeding words beginning with "A" are selected from the list of available storage and chained to the preceding entry.

An input item (say, EAT) is converted by use of the index to an initial table entry (in this case, 7). The table is entered (see Figure 4-1) and the Dictionary Address (477) used to perform a comparison (here, EAT with E). If the entries do not compare, the Next Table Address (14) is used to select the next table entry to be used, and the process is repeated. (In this case, EAT would be compared to EACH before being located on the third comparison.) If the input item is not found (i.e., the chain ends without a match), the end-of-chain code (*) is removed from that location, the address of an empty location is inserted to add another link to the chain, and the item is stored at the new location which is then marked as the end of the chain.

The average number of passes through the compare loop needed to locate a dictionary entry is:

$$C = \frac{C_L + 1}{2}$$

where: C is the number of compares

C_L is the average chain length

The time required by the compare loop is determined by the number of machine words in the entries being compared and by the amount of time needed to locate the next entry if there was a no-match condition. The dictionary can be partially pre-established (e.g., most frequent items at start of chains), to reduce the number of passes through the comparison loops and yield a more efficient method.

TABLE			DICTIONARY	
TABLE ADDRESS	NEXT TABLE ADDRESS	DICTIONARY ADDRESS	ADDRESS	TERM
1	2	480	475	CAT
2	4	485	476	DOG
3	8	483	477	E
4	6	487	478	EVEN
5	11	479	479	D
6	*	490	480	A
7	14	477	481	COW
8	9	475	482	DOT
9	10	481	483	C
10	*	486	484	EAT
11	12	489	485	ALL
12	13	476	486	CUR
13	*	482	487	ANY
14	15	488	488	EACH
15	16	484	489	DATE
16	*	478	490	AT

Figure 4-1. Chaining Tables

4.2.2 Binary Search Method

A second approach to dictionary processing is exemplified by scanning techniques, in particular the binary search. Scanning techniques compare an input item with each dictionary item one after another until matched. With N records in random order, an average of (N + 1)/2 items will have to be scanned. If an input item is not in the dictionary it requires N passes through the compare loop to determine that it is not there, so that except in trivial cases, sequential scanning of a dictionary to find a single item takes much too long.

However, the process time for the scanning technique can be reduced by maintaining the dictionary in sequence, and employing a binary search. If a dictionary of N items is stored in a random access memory with the items arranged so that their keys are in ascending (or descending) order, this technique may be used to locate an item in a time approximately proportional to $\log_2 N$. The binary search method locates an item by isolating the area in which the item should be found based on the sequence of the item keys. If the location of any one item is known, the direction of the search is determined by whether the desired key is higher or lower in sequence. The binary search technique begins by testing first the key of the item which is at the midpoint of the current search area. A comparison determines whether it is the desired item and, if it is not, the comparison specifies whether the next search for the desired item should be in the upper or lower half of the search area. This half is then bisected, and if necessary, the quarter of the area containing the sought item is determined. The bisection process continues until the item is located. (Figure 4-2 shows an instance of three such comparisons.) If the item is not in the dictionary, movement of dictionary items becomes necessary to insert the new item, and this is a time consuming operation. However, the binary search technique is efficient because at each comparison either the desired item is found or half of the remaining candidate items are eliminated from further consideration.

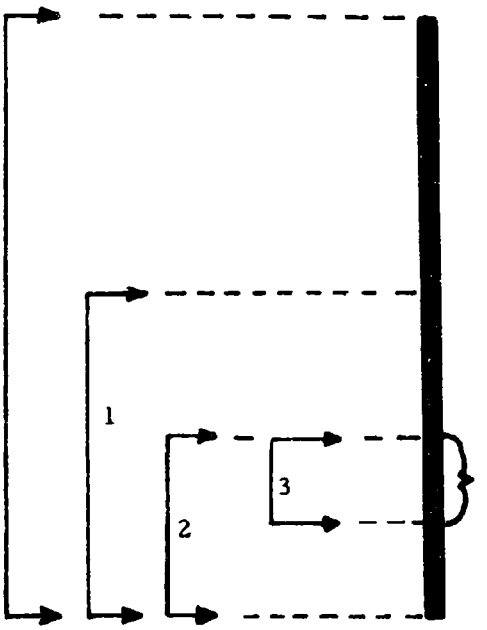


Figure 4-2. Binary Search

The binary search requires that the items be arranged in increasing (or decreasing) order in consecutive locations of a random access memory. Although the expected search time for this arrangement is relatively small, the time to alter the dictionary by adding or deleting items is proportional to N, because many items must be moved to make space for the new item. The maximum number of compares needed to locate an item or to determine that the entry is not in the dictionary block is:

$$C = \lceil \log_2 N \rceil + 1$$

where: $\lceil \log_2 N \rceil$ indicates the nearest integer greater than $\log_2 N$.

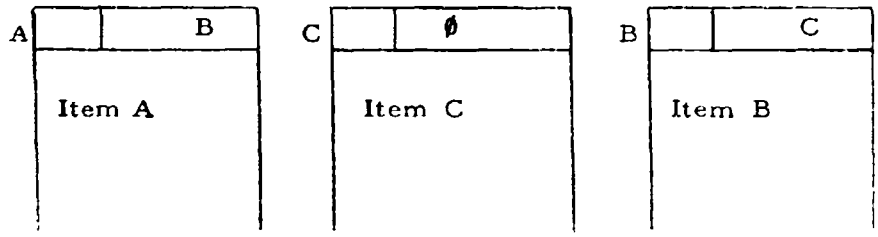
The average number of passes through the compare loop needed to locate any item is not much smaller than the maximum and can be expressed as:

$$C = 1/N \left(1 + \sum_{j=0}^{\lceil \log_2 N \rceil - 1} 2^j (j + 2) \right)$$

which is approximately equal to $C = \lceil \log_2 N \rceil$.

4.2.3 Directory-Chain Method

The directory-chain dictionary search method combines features of the preceding techniques. There is a significant variation in the method of selecting the proper table location and chain, and this procedure reduces the search and compare time considerably. In the following description of this method, chaining will be pictured as below:

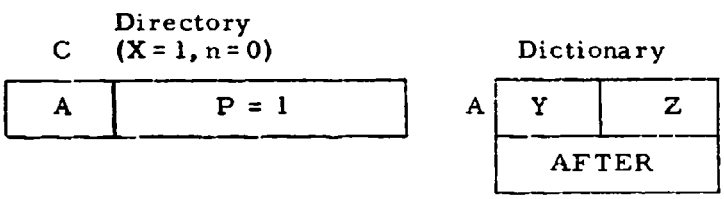


In this example, Item B occupies a chain position preceded by Item A and followed by Item C, which terminates this particular chain. In order to trace through the chain to extract its components in the proper order, a sequence of instructions is written which may proceed as follows: Obtain the address of Item A and extract that item. Examine the chaining position in A; this points to B and extract Item B, at the same time examining the chaining pointer. This indicates that C is the

next address. After the extraction of Item C, it is observed that the chain pointer is zero; there are therefore no more items in this particular chain.

For the directory-chain method, the complete table of addresses used in the straight chain method is replaced by an ordered directory of chain entry addresses. A binary search is used in the directory to reach the proper table location, where a pointer to the initial entry in each chain is maintained. A program parameter, Q, is specified as the maximum chain length. When the value Q is attained for any chain, e.g., chain i, an appropriate subroutine bisects this chain into subchains of length Q/2 and places the proper pointers and count of entries in locations i and $i + 1$ of the directory. Other directory entries may be pushed down to allow for this expansion if necessary. Thus, as the number of items in memory increases, the number of entry points to the chains also increases, and the number of searches within a chain never exceeds Q - 1. The speed of processing is therefore relatively independent of the input item distribution. (The "primed" modification to this method, which will be discussed later, would make efficient use of the input distribution.)

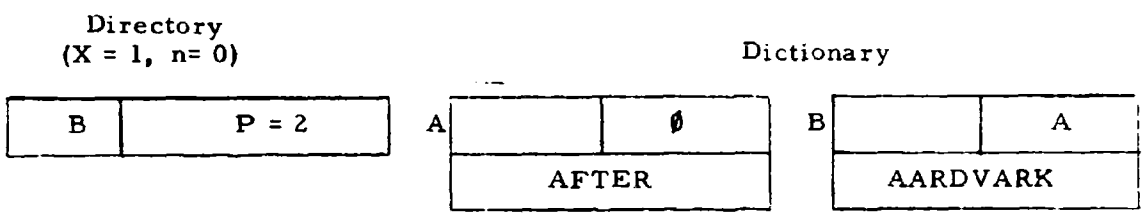
The following example should serve to clarify the general technique. The problem is to construct a dictionary of $N > Q$ entries, the first three input items being AFTER, AARDVARK, and ABACUS. The initial input item is AFTER, and at the end of its processing, the state of memory and of the directory is:



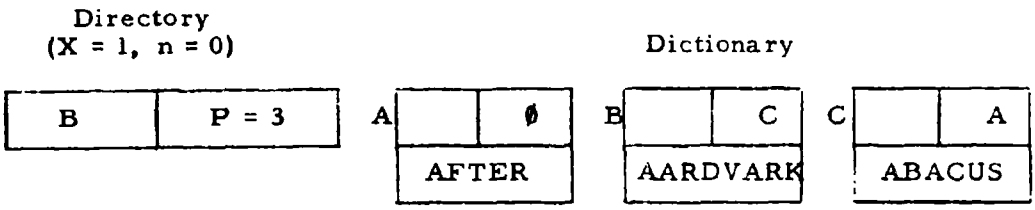
where:

- X is the number of chains currently in use in the directory. Here X = 1.
- n is the power of two used in the binary search, where $2^{n-1} < X \leq 2^n$. It is initially set to zero.
- C is the directory chain pointer. Here it points to the first (and at this point, only) chain of entries, beginning at location A.
- P is the number of entries in this chain. At all times, $P < Q$.
- Y is a free field and might be used to indicate the length of the entry.
- Z is either the address of the next entry in the chain or end-of-chain indicator.

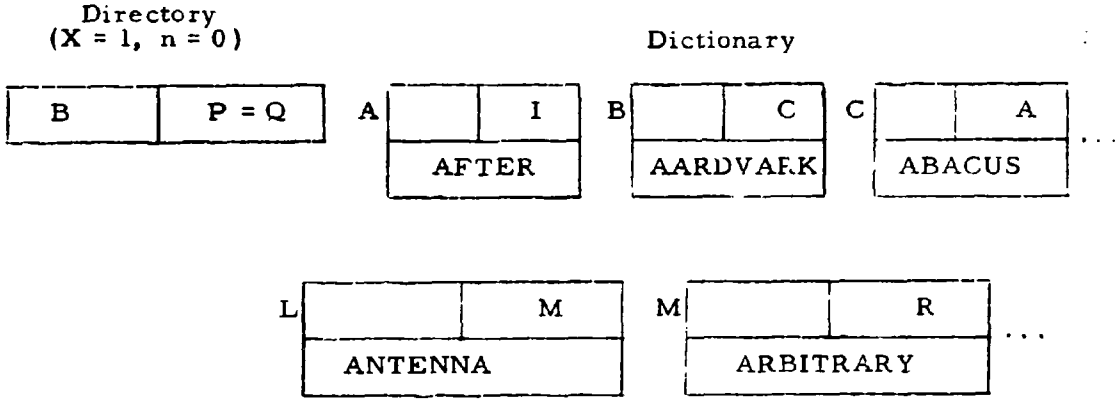
Let us suppose that AARDVARK is the second item in the input to be processed. A binary search is conducted using the directory and it is determined that the proper chain begins at location A. An inspection of the relative values of the two items establishes the fact that AARDVARK precedes AFTER. The former is therefore inserted in the chain prior to AFTER, and in this case, its location, B, is placed in the directory table to indicate the first item in the chain.



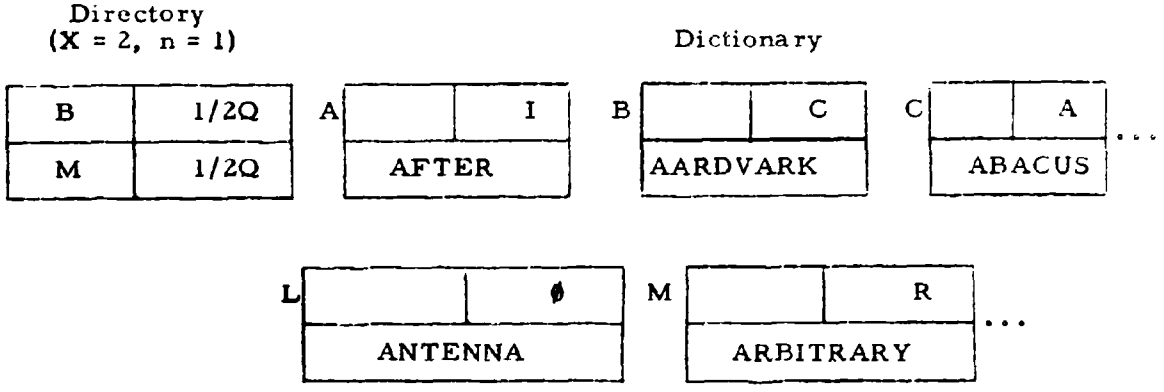
If ABACUS is the third input item, it must be inserted in the chain between AARDVARK and AFTER. The proper chain is found by a binary search using the directory, and the word inserted as shown below.



Eventually a point will be reached where P = Q. In the following diagram, I indicates the chain linkage to additional entries prior to ANTENNA in the first chain, and R indicates the chain linkage to additional entries following ARBITRARY.

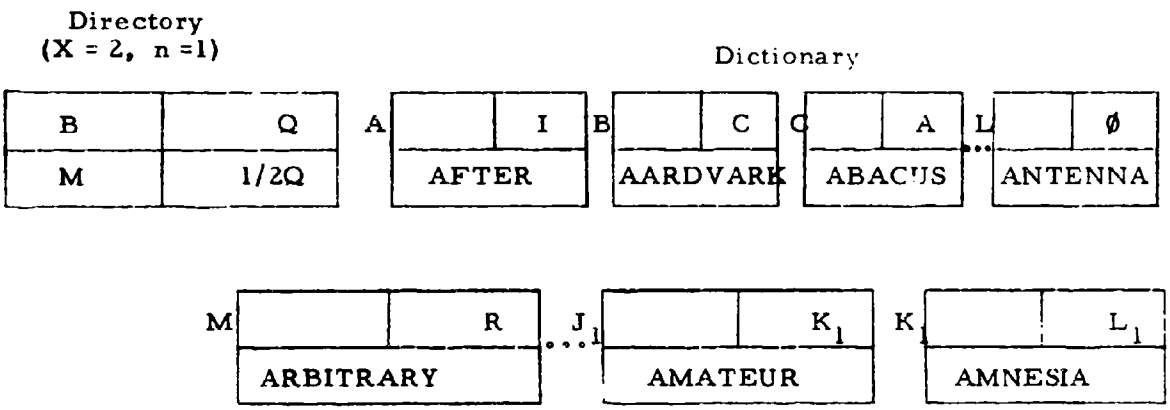


It is now advisable that the chain be split. Suppose that entry L is in the 1/2 Q^{th} position in the chain. The directory is expanded in the following fashion:

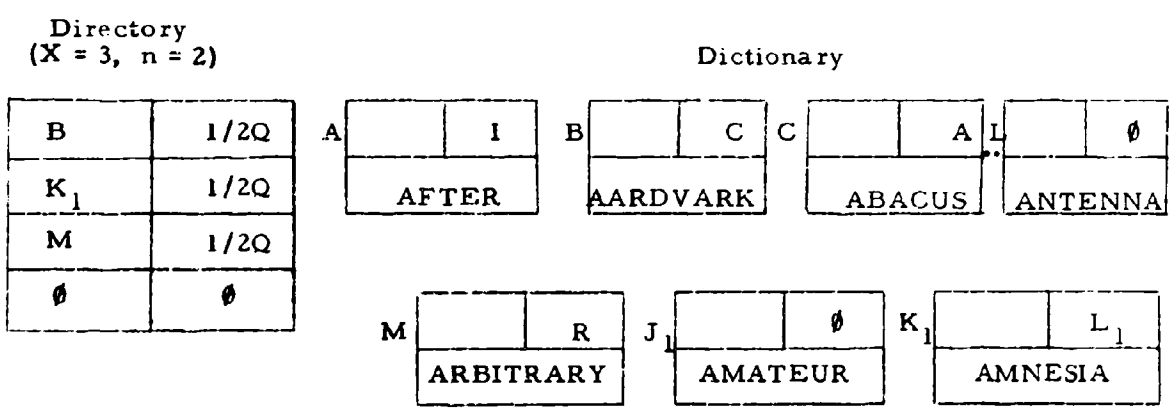


An end-of-chain indicator has been inserted in the chaining portion of entry L, and the entry M has been indicated as the starting point of a new chain. X is increased by one (X=2) since there is an additional entry in the directory, and since $X > 2^0 = 1$, n is increased by one (n = 1) for an enlarged binary search.

Let us assume that 1/2 Q additional entries have now been processed which fall in the range of values between AARDVARK and ANTENNA. Suppose furthermore that AMATEUR and AMNESIA are now the medial entries in the chain headed by AARDVARK, and that J_1, K_1, and L_1 exist in the chain somewhere between B and L.



The B chain must again be split since the entry count has reached Q. Chain and directory adjustments are made:

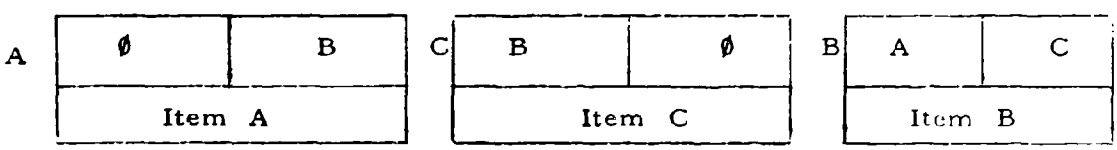


Again $X > 2^n$, and n is incremented by 1 (n = 2) so that $2^n = 4$ for the binary search. Since the entry count for chain M has not reached Q, the table entry is merely pushed down to make room for the expansion above. This process is continued until all available memory is in use. If n expands beyond the boundaries of storage allocated for the directory, the parameter Q may be increased or an entrance to the output section may be effected.

There are two types of modifications to the directory-chain dictionary search technique which could improve the overall dictionary construction operation. The first modification consists of "priming" the heads of initial chains with those words whose distribution in the input is frequent in occurrence. For example, it has been observed in several large samples of narrative English text that approximately 35% of the tokens (total words) are accounted for by 40 types (unique words). Also, for the same text samples, it has been noted that 200 types account for 45% of the tokens. Thus, if the directory-chain technique was primed with these 200 words at the heads of the chains for each pass, the processing time could be reduced significantly. Elimination or replacement of these chain heads could be easily accomplished either at the start of the processing or at the end of each pass of the input.

The other modification which could be made is to employ bi-directional chaining using a floating pointer. As in the other chain techniques, all available memory is utilized to provide the opportunity to process the greatest number of items at one time, and the first N bits of an item are again used to identify the table location pertaining to the proper sub-group of items.

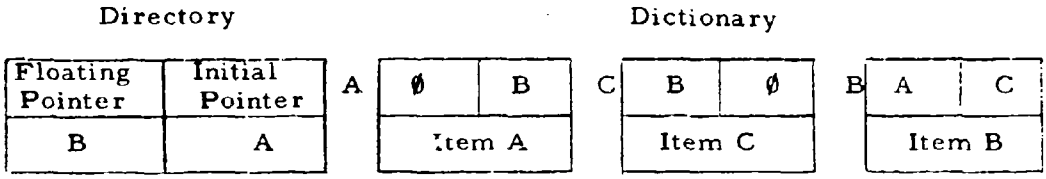
In bi-directional chaining the following convention will be used:



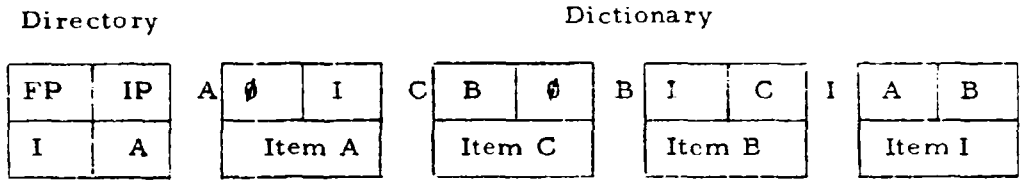
Here, the chain may be traced in either of two directions, and Item B occupies an intermediate position relative to Items A and C as terminal entries on their respective ends of the chain.

To assist the bi-directional chaining a floating pointer is used. The purpose of this pointer is to decrease the number of inspections in each chain by pointing to a second location in the chain which is used for searching as well as to the beginning of the chain which is used for reading. On the basis of the relationship between the new entry and the entry indicated by the floating point, the examination of other items in the chain may proceed on either of the paths provided by the bi-directional chaining.

To illustrate bi-directional chaining, suppose a new entry Item I is obtained such that Item A < Item I < Item B, the following would be the structure before the processing of Item I:



and after the processing of Item I:



It is evident that the pointer in A has been altered to include Item I, as has the pointer in B. Similarly, I now contains pointers to both A and B.

If the floating pointer is treated as a mid-point position, the sub-group chain is effectively cut in half. For a random distribution, this method eliminates the need for approximately 50% of the inspections required in the straight chain method, and for an updating operation a time-saving of approximately 50% is also realized. However, for a file or group inversion operation there is a disadvantage to the technique as compared to the straight chain method. A possible solution to this would be the utilization of the initial pointer to ascertain the more favorable inspection point. This would increase slightly the operational time for other types of distribution where the floating pointer is the more favorable entrance point to the chain, but a general time-saving is realized in the former case.

For actual word distributions in text, a better use of the floating pointer can be made. This better usage is achieved when the pointer is not restricted to the median position but is allowed to float in the direction of chain additions. For a random distribution of items, this technique is essentially similar to the above floating pointer method; however, for either a file inversion or an up-dating operation, a saving in operational time is achieved since the floating pointer will tend to anticipate the direction of the chain motion. This method represents the best version of the chaining technique studied so far.

A further potential improvement consists of breaking chains at the mid-points of the distributions of items in the chains rather than at the mid-points of the chains themselves. This should tend to concentrate the frequent words in short chains while infrequent words remain in longer chains. It, of course, requires more bookkeeping and may not prove to be advantageous.

Further work will be done on these methods prior to a selection being made for purposes of program design work. The final choice will depend not only on the theoretical efficiency of the method but also on the actual machine cylces used in the inner loop and the ease of coding and maintaining it.

Section 5

GLOSSARY

This Glossary is intended to serve as an aid to understanding some of the specialized terminology used in this report.

character	a decimal digit zero to nine, or a letter A to Z, either capital or lower case, a punctuation symbol, a blank or any other symbol, which a machine may read, store, or write.
delimiters	a specified set of characters, a member or members of which determine the boundaries of an item.
item	a continuous sequence of one or more characters; for example, in defining a word an item is bounded by delimiters; for a n-gram an item is determined by length.
item token	the occurrence(s) of a unique item (type) in the input data; the type "for" had a token occurrence of six times.
item type	a unique item which could occur in the input data; types could be "for", "Tom", "with", etc.
key	the primary identifier of an item such as initial character or character group, by which that item can be identified or searched.
module	the set of instructions necessary to direct the computer to execute a well defined mathematical or logical operation; a subunit of the basic program.
tag	a unit of information, whose composition or location differs from that of other members of the data set so that it can be used as a marker or label.

word pair — two sequential words (items) separated by one or more delimiters; in the three word phrase "solid logic technology", word pairs are "solid logic" and "logic technology".

SUPPLEMENTARY

INFORMATION

NOTICE OF CHANGES IN CLASSIFICATION, DISTRIBUTION AND AVAILABILITY

69-18 15 SEPTEMBER 1969

AD-485 188 IBM Federal Systems Div., Gaithersburg, Md. Annual progress rept. Apr 66 Contract Nonr-4456(00)	No Foreign without approval of Office of Naval Research, Attn: Information Systems Branch, Washington, D. C.	No limitation	ONR, D/N ltr, 18 Jun 69